Master's Thesis

# STCP: Receiver-agnostic Communication Enabled by Space-Time Cloud Pointers

YeoCheon Yun

Department of Electrical and Computer Engineering
(Computer Engineering)

Graduate School of UNIST

2016

# STCP: Receiver-agnostic Communication Enabled by Space-Time Cloud Pointers

YeoCheon Yun

Department of Electrical and Computer Engineering
(Computer Engineering)

Graduate School of UNIST

# STCP: Receiver-agnostic Communication Enabled by Space-Time Cloud Pointers

A thesis
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

YeoCheon Yun

07. 07. 2016
Approved by

_____

Advisor
Kyunghan Lee

# STCP: Receiver-agnostic Communication Enabled by Space-Time Cloud Pointers

YeoCheon Yun

This certifies that the thesis of YeoCheon Yun is approved.

07. 07. 2016

signature

_____

Advisor: Kyunghan Lee

signature

_____

Prof. Changhee Joo

signature

_____

Prof. Hyoil Kim

# ABSTRACT

During the last decade, mobile communication technologies have rapidly evolved and ubiquitous network connectivity is nearly achieved. However, we observe that there are critical situations where none of the existing mobile communication technologies is usable. Such situations are often found when messages need to be delivered to arbitrary persons or devices that are located in a specific space at a specific time. For instance at a disaster scene, current communication methods are incapable of delivering messages of a rescuer to the group of people at a specific area even when their cellular connections are alive because the rescuer cannot specify the receivers of the messages. We name this as receiver-unknown problem and propose a viable solution called *SpaceMessaging*. SpaceMessaging adopts the idea of *Post-it* by which we casually deliver our messages to a person who happens to visit a location at a random moment. To enable SpaceMessaging, we realize the concept of posting messages to a space by implementing *cloud-pointers* at a cloud server to which messages can be posted and from which messages can fetched by arbitrary mobile devices that are located at that space. Our Android-based prototype of SpaceMessaging, which particularly maps a cloud-pointer to a WiFi signal fingerprint captured from mobile devices, demonstrates that it first allows mobile devices to deliver messages to a specific space and to listen to the messages of a specific space in a highly accurate manner (with more than 90% of *Recall*).

# Contents

# List of Figures

# 1 Introduction

The cellular network services have persistently expanded their coverages in every nation during the recent decades. According to [17], the carriers who were providing the best LTE network coverage as of Feb. 2016 in south Korea and USA showed 97% and 81% coverage, respectively. Also, [17] revealed that most LTE networks in the world were able to provide more than 10 Mbps of downlink speed, which shows its peak at 37 Mbps in Singapore. This nearperfect coverage with fast link-speed is allowing people to communicate with anyone, anywhere, at anytime.

However, we found that there are unusual but important communication needs that cannot be met even with those cellular network technologies. An immediate example of such needs is observable at a disaster scene where message delivery is vital toward unidentified people waiting to be rescued, whose cellular connections are all alive. Intriguingly, the reason of communication unavailability in such a situation is not because of insufficient coverage or weak signal strength, but because of lack of knowledge on the identities of the candidate receivers. In most conventional communication systems, we have taken it for granted that the receiver's identity such as its IP address or its phone number is given a priori before making a connection. However, our example scenario clarifies that we sometimes need communication even when such information is unavailable. This may be the reason why we still heavily rely on old-fashioned megaphones to deliver information in a disaster scene, although that method quickly shows its limitations when trying to deliver more complicated information such as evacuation maps or manuals to operate escape gears. A cellular or WiFi broadcast can be of a modern solution, but their inability for specifying a target area and the resulted deluge of irrelevant messages critically restrict their efficacy. This observation made us to devise a new way of communication that does not require any receiver identity, but is able to deliver messages to any receiver at a specific space (and time). We call such a concept of communication, *SpaceMessaging*.

The usefulness of SpaceMessaging is more widely observed in our daily lives. For instance, a mom who should go out for work in the early morning while her children are still sleeping puts a *post-it* note on her refrigerator to let her children know what to have for breakfast. The mom may know how to send a text message, a tweet, or a facebook message, but she chose to put a post-it note. We know why. It is because the post-it note does not unwillingly disturb her children from sleeping, and is not easily overlooked if her children really wants to have something for breakfast. In such a tricky situation where a conventional messaging is not of the best to use, SpaceMessaging is particularly useful and can replace the use of post-it notes by allowing messages to be sent toward the space surrounding the refrigerator. It is even possible for SpaceMessaging to make the messages expired right after the morning time. Also, compared to typical post-it messages, the messages of SpaceMessaging can be far richer by having various types of data such as images, voice data, multimedia data, and hyper-links embedded. This implies that she can even deliver a video recording to her children on how to heat up and prepare the breakfast, which is far beyond the ability of post-it notes.

Given the projected benefits of SpaceMessaging, an immediate question would be if it is possible to realize a communication technique which precisely delivers a message toward a specific space, especially when GPS (Global Positioning System) is not reachable such as at indoor spaces. To answer this question, we propose a SpaceMessaging system that implements perspace virtual data storage named *cloud-pointers* on a cloud server. In our system implemented over Android phones and Linux servers, each cloud-pointer gets a mapping to a corresponding physical location. The mapping between cloud-pointers and physical locations can be made from various information sources available at mobile devices, but in this work we opt to use only WiFi signal fingerprints that are easy to obtain. Once a mapping is made, sending a message to a cloud-pointer is possible by embedding the on-site signal fingerprint to the message, which can be interpreted in the server to store the message to the corresponding cloud-pointer. Receiving a message from a cloud-pointer becomes possible by sending a query embedding the on-site fingerprint, with which the server will check if there is a message stored in the corresponding cloud-pointer and return it if there is any. For the evaluation of our SpaceMessaging system, we carried out extensive measurement studies as well as operational studies at a building of about 1200 square-meters in which about 450 wireless access points are observable in total. From these experiments, we confirm that it is possible to precisely specify a space whose radius is of a few meters (e.g., 3, 10, and 20 meters) without the help of an indoor localization system when the vector space distance between WiFi signal fingerprints is judiciously chosen. With this, we further confirm that posting a message to a specific space and fetching a message from a space by the implementation of SpaceMessaging are both possible with very high accuracy (over 90%

*Recall*[1]).

# 2 Related Work

We provide related work of SpaceMessaging in three different perspectives.

**Geographical multicasting:** There have been a concept of geographical broadcasting or more specifically multicasting whose main focus was mostly given toward routing. Geocast [15] suggested the first networking architecture for geographical routing and proposed geographical addressing of network entities. Geocast envisioned various networking services including geographical messaging, geographical advertising, and geographical service differentiation. Although the vision of Geocast was attractive, its complication over routing and addressing hindered it from being popularized on the Internet. Unlike the Internet, the idea of Geocast has been rather successful in ad-hoc networks. With the new design of packets embedding GPS coordinates of destination areas, mobile ad-hoc networks involving military, vehicular networking, and sensor networking applications have reinvented various versions of geographical multicasting as in [3,9,21,22]. However, the extreme success of cellular networks over wireless multi-hop networks made them out-dated and unavailable for smartphones. As of now, there are only few realizations of the concept of geographical multicasting, which are cell-broadcasting [2] and social medias with location tagging [18]. Cell-broadcasting is a naive implementation of geographical broadcasting whose broadcast scope is bounded by a set of cell towers and their associated cellular devices. A more sophisticated delivery is beyond the scope of cell-broadcasting. Social medias with location tagging are also far from the gist of geographical multicasting because a posting with a location tag is delivered to all of the connected people (so called friends) no matter where they are rather than being delivered to the anonymous people at the location. **Delivery time reservation:** In order to deliver a message to a specific space during a predefined time period, the function of delivery time scheduling is essential. Scheduled SMS has been implemented by various cellular network carriers and service providers in the world such as SKT or KT in South Korea, and services such as *Oh, Don't Forget* [6]. A recent approach of message scheduling is available on the market, which is by smartphone applications such as SMS Later [1] and Autotext [7]. These applications enable the scheduled delivery using the computing resources of a smartphone without relying on the messaging servers of cellular providers.

**Localization techniques:** Most conventional geographical multicasting techniques have considered outdoor scenarios because indoor localization has long been considered as a challenging task. However, recent advances in indoor localization system have shown that highly precise indoor positioning [25] as well as indoor navigation [19] are practically achievable without putting too much cost (e.g., laboring or equipment cost) [23]. These advances are being made by hybrid approaches combining several of wireless signal fingerprinting [8,11,16], signal propagation modeling [4], sensor data fingerprinting [23], dead-reckoning [20], map-matching [12], and vision assistance [10]. However in our implementation of SpaceMessaging, we presume that none of these indoor localization services is available given the reality that only a small portion of buildings in the world provides such services. To this end, instead of relying on an indoor localization service, we adopt the idea of unsupervised clustering of WiFi signal fingerprints that are crowd-sourced at a building [8] and realize cloud-pointers by establishing a mapping with the detected clusters without having the notion of physical indoor coordinates.

# 3 System Design

## 3.1 Overview

SpaceMessaging aims at enabling people to talk to a space with the assistance of a cloud server system. In order to strictly rule out the necessity of identities of the mobile devices being located at a space, SpaceMessaging works with two types of servers: cloud-pointer server and cloud-message server. The cloud-pointer server manages *cloud-pointers*, each of which is mapped with a range of sensor information collected from mobile devices being at a space. Each cloud-pointer stores the address of a message storage on the cloud-message server. Note that the

[1] The formal definition of recall is given in the evaluation section

types of sensor information used to identify a space may be various even including GPS coordinates when available, but we here focus on WiFi signal fingerprint, the signal strength vector of observable WiFi access points. A new cloud-pointer is created upon the request from a mobile device being at a space of a differentiated WiFi fingerprint that is determined to be distinct from the fingerprints which are clustered to existing cloud-pointers. We will discuss how the fingerprints can be clustered in the next section. Once a cloud-pointer is created, its corresponding message storage becomes a virtual bulletin board or a data warehouse for the mobile devices being at the space. For each message storage connected to a cloud-pointer, the cloud-message server manages the list of device identities (e.g., cellphone numbers or email addresses) who have ever accessed the storage. If it is required by a policy for the message server to deliver messages even to the mobile devices who had accessed the storage (e.g., by posting a message) but are currently out of the corresponding space, the server utilizes the list and can push the messages out to those devices. In a similar manner, we can design forwarding policies including "to all devices who have ever visited," or "to the devices who are currently being at the space," or more complexly "to the devices who are being at the space and to the devices who were at the space when I was there." The forwarding policies can be further elaborated to support secure and private delivery, which is beyond the scope of this paper and can be of our future work.

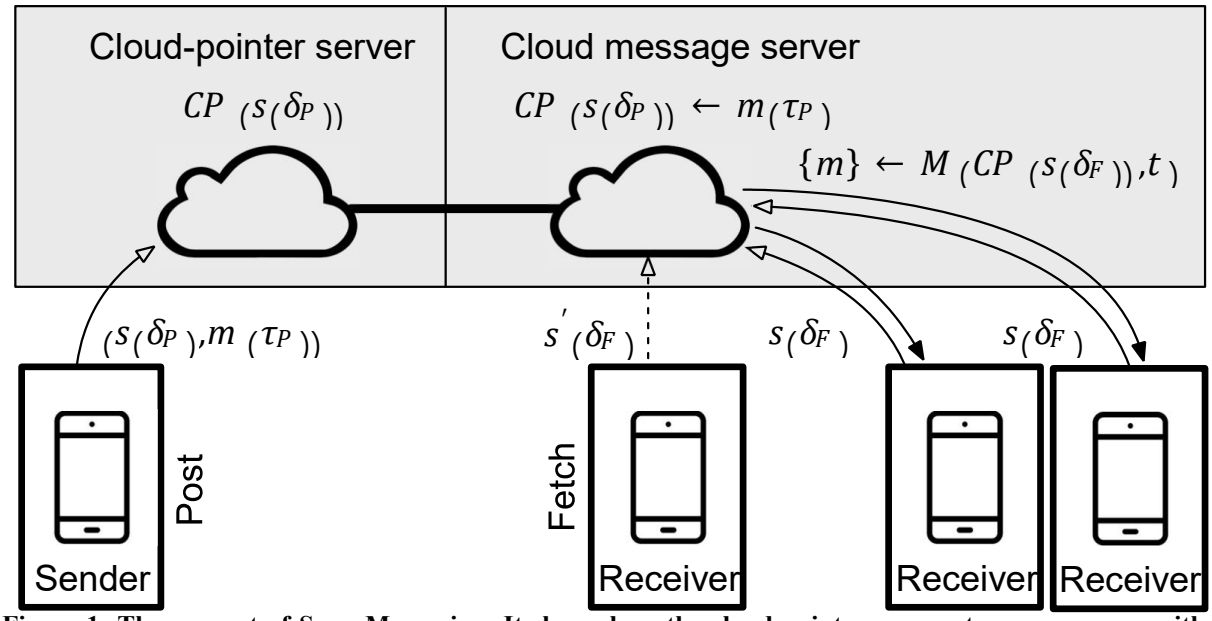



**Figure 1: The concept of SpaceMessaging: It shows how the cloud-pointer server stores a message with signal fingerprint sent from a sender and how a set of receivers fetch messages stored in the corresponding cloud message server for their signal fingerprints.**

Figure 1 shows the concept and basic operations of SpaceMessaging. As the figure suggests, in order for a message sending device to post a message $m$ to a target space, we let the device initiate a message at the space, which embeds the WiFi signal fingerprint $s$ captured at the space. When receiving a request for posting a message, the cloud-pointer server identifies the corresponding cloud-pointer for $s$, which is defined by $CP(s)$ and stores the message at its message storage. We simply denote this operation as $CP(s) \leftarrow m$. From this conception, we find that there may be users who need to post a message not only to a cloud-pointer but also to nearby cloud-pointers. To accommodate this need, we add a notion of the radius for posting denoted by $\delta_P$ for a WiFi fingerprint and make a fingerprint to be embedded in a message as $s(\delta_P)$. From this extension, $CP(s(\delta_P))$ returns all the cloud-pointers that are within the radius of $\delta_P$ from $s$. For user friendliness, $\delta_P$ can be presented to users in the unit of meters, but internally it will be of a measure determining the vector space distance between WiFi fingerprints. Another simple extension can be made for the available time period of a message as $\tau_P$. This limits the message $m(\tau_P)$ to be only visible during the period specified in $\tau_P$. Having this included, the operation at the cloud servers becomes $CP(s(\delta_P)) \leftarrow m(\tau_P)$.

In order for a message receiving device to fetch messages from message storages, the device sends a query embedding its signal fingerprint $s$ in the form of $s(\delta_F)$ where $\delta_F$ denotes the radius of nearby space to look up in the cloud-pointer server. When the cloud-pointer server finds a set of cloud-pointers satisfying $s(\delta_F)$, the server returns those cloud-pointers and asks the cloud message server to deliver the set of messages $\{m\}$ in the

corresponding message storages. Note that when message storages return messages, they check the current time $t$ and filter out those messages that are not intended to be delivered at $t$. This series of operations is denoted by $\{m\} \leftarrow M(CP(s(\delta_F)),t)$, where $M$ stands for the function of message storage look-up and $\{m\}$ denote the set of messages acquired from multiple cloud-pointers. Note that as in Figure 1, when there is no matching cloud-pointer for a signal fingerprint (e.g., $s^0$), clearly no message is fetched.

### 3.2 Extension

The message delivery capability of SpaceMessaging is not limited to the delivery of a text message. The message can be of any format including documents, audio files, video files, and even sensory information. To this end, we extend the scope of a message to include *commands* that are equivalent to remote function calls. For instance, if we embed *launch a map application* command with a *navigation path* in a message, the receiver of this message will present the navigation path to its user. In a similar manner, if we embed *callback* command with *callback address* such as skype ID or cellphone number, the receiver of that message will give an automated call to the sender. If the callback address is given as a session ID of a conference call, the receivers will join the conference call session and start a many-to-many call. This extension immediately gives a solution to the disaster scene exemplified in the introduction. We will discuss more about the implementation of this extension in Section 5.

## 4 Algorithms

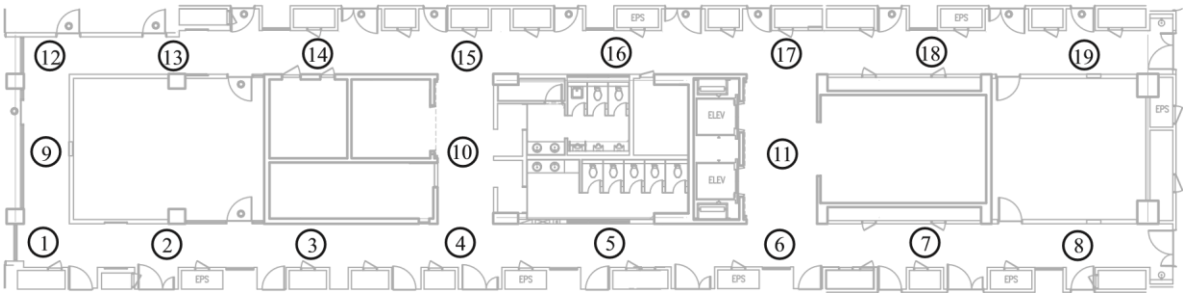



**Figure 2: The spaces where we have collected WiFi signal fingerprints are indexed from 1 to 19. The spacing between the centers of neighboring spaces is about 7 meters.**

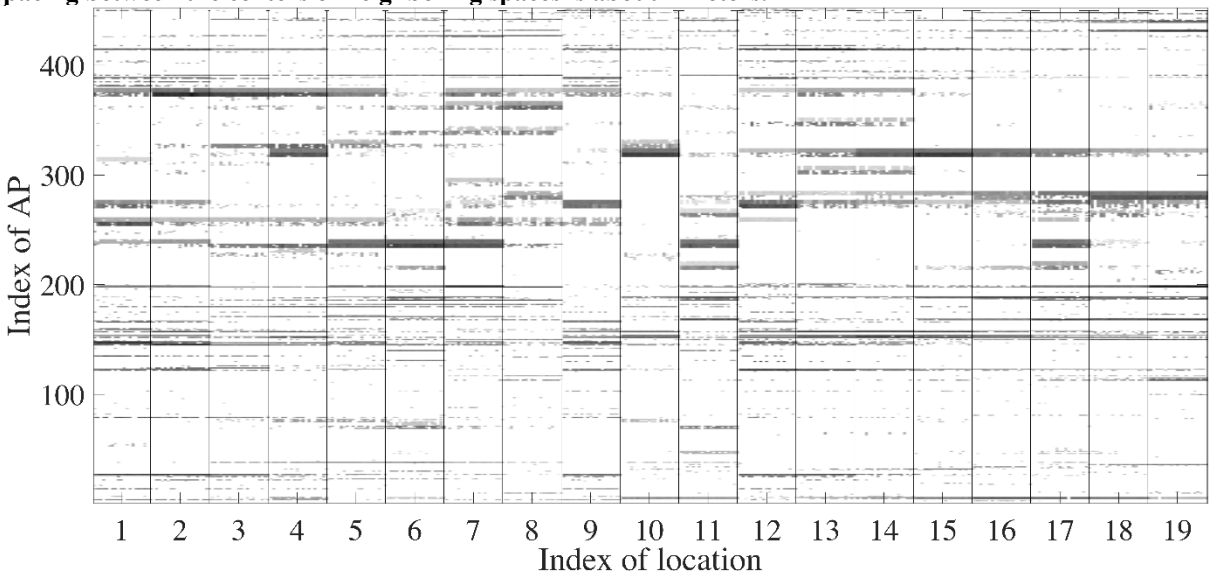



**Figure 3: Sample fingerprints from 19 indexed spaces. Each space presents 20 samples. In each sample,**

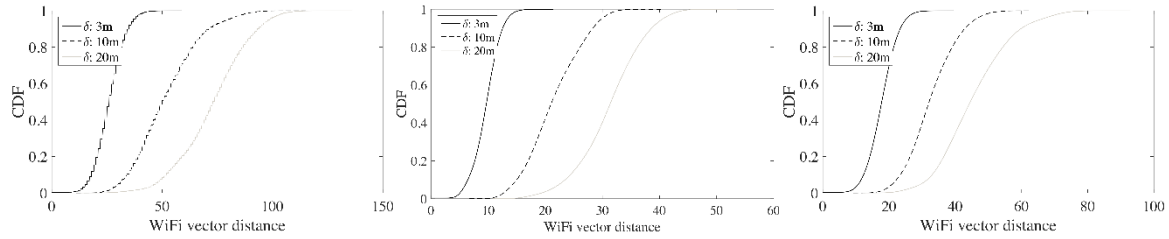
**each indexed access point is mapped with a color (black to white) based on the observed signal strength.**

In this section, we formally describe our algorithms on how the fingerprints are processed and how the cloud-pointers are created from the clusters of fingerprints. To develop efficient algorithms, we performed our own measurement study at a campus building of about 1200 square-meters, whose floor plan is presented in Figure 2. We intensively sampled WiFi signal fingerprints around 19 spaces marked in the floor plan. Note that because the goal of SpaceMessaging is to deliver messages to some representative spaces (which we call by point of interest (PoI)), it may be enough for SpaceMessaging to separate a finite number of spaces instead of pursuing a sub-meter scale localization accuracy. From this measurement study, we find about 450 access points that are distinct to each other. Figure 3 visualizes 20 fingerprints from each space out of 19 spaces. Each fingerprint depicted as a vertical strip shows the observed signal strength of 450 access points by the color. When the color is close to black, the signal strength is maxed (at about -33 dBm) and is close to white the signal strength becomes weak (with the weakest at -102 dBm). The pure white color means that the corresponding access point is not observed in that fingerprint. As shown in the figure, there are popular access points over most of the spaces while some of the access points are only visible at few spaces. One important observation here is that nearby spaces have a certain level of similarity which becomes the basis for fingerprint clustering as well as the radii of posting and fetching ($\delta_P$, $\delta_F$).

## 4.1 Fingerprint Processing

In order to cluster fingerprints, it is essential to define a distance metric that measures the closeness between fingerprints. In SpaceMessaging, we use the vector space distance as the distance between fingerprints where a fingerprint $s$ forms a vector $x$ using the following equation.

$$x_i = \alpha \cdot I(s_i) + (1 - \alpha) \cdot V(s_i),$$




(a) α = 1 (Full weight to existence) (b) α = 0 (Full weight to RSSI) (c) α = 0.5 (Full weight to RSSI)

**Figure 4: CDFs of signal vector distances measured by Euclidean norm (2-norm) between fingerprints collected inside two physical radii: [0,3] meters, [3,10] meters, and [10,20] meters. A signal vector is formed from a fingerprint with (a)α= 1, (b)α= 0 and (c)α= 0.5**

where $x_i$ and $s_i$ denote the $i$-th dimension value of the signal vector and the signal fingerprint for the access point of index $i$. $I(s_i)$ and $V(s_i)$ denote the existence of the access point $i$ and the normalized signal strength (RSSI: received signal strength) for the access point $i$. Note that $I(s_i)$ is either of 0 or 1 and $V(s_i)$ is a certain real number between 0 and 1. $\alpha \in [0,1]$ is a weighting factor balancing the importance of existence and signal strength in the vector formation. By knowing the required number of dimensions for $x$ in a building through a very rough site survey, we can easily convert a fingerprint in the cloud-pointer server to a signal vector. Figure 4 presents CDFs (cumulative density function) of signal vector distances measured by Euclidean norm between fingerprints. Each CDF is drawn by inspecting the distances of all pairs of fingerprints which are collected between two radii such as [0,3] meters, [3,10] meters, and [10, 20] meters. Figure 4 (a), (b), and (c) shows the CDFs for different *alpha* values: 1, 0, and 0.5 to see the impact of existence and RSSI of access points. As Figure 4 verifies the vector space distance between fingerprints gets larger as the physical space gets more separated. Also, Figure 4 confirms that when giving its full weight to RSSI in constructing a signal vector from a fingerprint, the fingerprints are more clearly separated in the vector space, by showing relatively wider spacing between CDFs in Figure 4 (b).

## 4.2 Vector Clustering

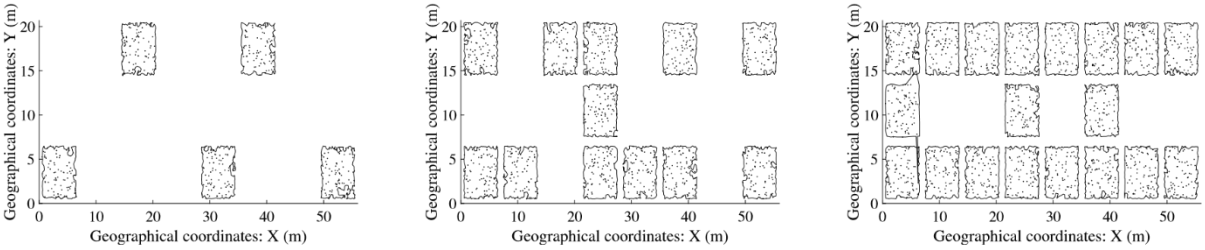
Once signal vectors begin to be collected in the cloud-pointer server, we can run a clustering algorithm for fingerprints to create cloud-pointers. There exist a huge number of clustering techniques. In SpaceMessaging, we choose Ward's clustering (WC) [24] as our default clustering method. Ward's clustering is one of agglomerative hierarchical clustering techniques where agglomerative hierarchical clustering means a bottom-up approach of clustering data points that repeatedly merges available clusters initiated from all individual data points until a single top level cluster is made. Once Ward's clustering completes to cluster input data points, by its nature, a cluster tree is formed whose bottom leaves are all individual data points. This tree is useful to find out an arbitrary number of clusters since a simple track-back from the top to the bottom gives an increasing number of clusters from 1 to the number of input data points. Stopping at the middle of a track-back process gives a corresponding number of clusters which is the outcome of a horizontal cut of the cluster tree. Ward's clustering shows a different clustering result compared with $K-means$ clustering [14], even when targeting to create the same number of clusters. Intuitively, Ward's clustering gives a more consistent clustering result as the target number of clusters decrease, by the nature of its hierarchy in the cluster tree. For instance, the clustered data points from $k$ clusters stay all together when $k-1$ clusters are formed in Ward's clustering, but this property does not hold in $K-means$ clustering and the data points are regrouped at every level of clustering.

A general agglomerative hierarchical clustering including Ward's clustering works in the following three steps: 1) Let each data point be a cluster and calculate distances among them using a distance metric, 2) Find the closest clusters, merge them and update the distances among clusters, 3) Repeat the merge process until the remaining cluster becomes one (i.e., all clusters are hierarchically merged into a single cluster). The distance metric to measure the cluster distance for Ward's clustering $D_w$ is defined as follows:

$$D_w(C_i, C_j) = \sum_{x \in C_i} ||x - r_i||^2 + \sum_{x \in C_j} ||x - r_j||^2 - \sum_{x \in C_{ij}} ||x - r_{ij}||^2 \quad , \tag{1}$$

where $x$ is an individual data point (i.e., a signal vector in SpaceMessaging) and $r_i, r_j, r_{ij}$ denote the centroids of clusters $C_i, C_j, C_{ij}$, respectively. $\|\cdot\|$ denotes the length (i.e., Euclidean norm) of a signal vector. Note that $C_{ij}$ is the virtual union of clusters $C_i$ and $C_j$.

## 4.3 Cloud-pointer Generation




(a) $k_{\mathrm{PoI}} = 5$ (b) $k_{\mathrm{PoI}} = 12$ (c) $k_{\mathrm{PoI}} = 19$

**Figure 5: Signal fingerprints are clustered with (a) kPoI = 5, (b) kPoI = 12, and (c) kPoI = 19 by Ward's clustering. Each signal fingerprint used for clustering is depicted as a dot at its physical coordinates we recorded as ground truth. Each cluster is** depicted with its boundary.

Given the notion of a cloud-pointer that is designed to act as a virtual bulletin board at a space, we need the coordinates of a cloud-pointer in the signal vector space so that signal fingerprints that are determined to be close can post and fetch messages. We set up the coordinates of cloud-pointers in a building by crowd-sourcing signal fingerprints from the mobile devices in the building during a short training period (e.g., a day) or simply by letting a user visit points of interest in the building during a few hours without tagging spaces. From either of fingerprint collection process, we are able to obtain raw fingerprint database. We put this database as the input of Ward's clustering and run the clustering to find out $k_{\mathrm{PoI}}$ clusters, where $k_{\mathrm{PoI}}$ denotes the number of point of interests (i.e.,

representative spaces) in the building. Through the measurement study mentioned earlier in this section, we empirically verified that Ward's clustering is capable of finding PoIs where many similar signal fingerprints are collected. Therefore, as long as the raw fingerprint database captures the uneven distribution of signal fingerprints that naturally reflect the popular spaces of the building where people stay longer than other spaces (e.g., corridors that have little attraction), we can find out clusters that we are interested in. For instance, as shown in Figure 5 (a), (b), and (c), by providing a reasonable number of PoIs: 5, 12, and 19 in a building, we get reasonable reasonable clustering with almost no exceptions. Each cluster has its centroid which is calculated from the signal vectors included in the cluster. The centroids of clusters in the signal vector space map to cloud-pointers of the corresponding points of interest.

### 4.4 Posting and Fetching

Once the centroids of cloud-pointers are found in the training period, we are able to post a message with $s(\delta_P)$ using the observations made at Figure 4. For instance, if a user wants to post a message to all the cloud-pointers within 3 meter distance from her current position, she can set $\delta_P$: $3m$ and ask the cloud-pointer server to find out all relevant cloud-pointers satisfying the vector space distance corresponding to $3m$ from the fingerprint $s$. Then, all the message storages connected to those cloud-pointers saves the message from her. In the exactly opposite manner, a user can fetch messages from the nearby message storages determined to satisfy the vector distance $\delta_F$ from the signal fingerprint $s$ captured at the user's space. The posting process and the fetching process sometimes incur errors because of the instability involved in signal fingerprints. We find that either of posting and fetching experience types of events *TP*, *TN*, *FP*, and *FN* where *T, F, P* and *N* stand for true, false, positive and negative. We will give a formal definition of each event and will evaluate the performance of SpaceMessaging using these notions in Section 6.

### 4.5 Complexity of Posting and Fetching

For posting and fetching, our system compares vector distances between the user-sent fingerprint and the fingerprints of all centroids of the cloud-pointers with the specified distance thresholds for posting and fetching (i.e., $\delta_P$ and $\delta_F$). The computational complexity involved in this process consists is in the order of $O(NC)$ where $N$ and $C$ represent the dimensional size of the fingerprint vector and the number of clusters to compare with. We can keep $C$ small enough in practice by tagging all the clusters in the server with the building ID and by identifying the building ID to bring up the candidate clusters through a mapping between MAC addresses of the observed access points and the building IDs. Note that once the building ID is determined, the vector dimension $N$ can also be reduced by ruling out the access points that were not observed in that building. Give small enough $N$ and $C$, the computation of $O(NC)$ is manageable in the server. However, there is another need for optimizing the computational complexity in the server, which comes from the redundancy between message fetch queries made by each user device. Since each device is designed to generate a query regularly, there are cases where the embedded signal fingerprints in the subsequent queries are nearly the same. In order to mitigate the computational burden from this redundant query, we provide a simple vector comparison logic in the user device so that each device can autonomously suppress sending out redundant queries toward the server.

## 5 Implementations

In this section, we describe how our system is implemented in detail. Our implementation consists of two different platforms, Android and Linux. Android implementation is for user devices and deals with user-side functions that involve both message posting and fetching operations. Linux implementation is to operate cloud-pointer server and cloud message server that receive, store, and deliver messages. Each implementation is explained in the following subsections separately.

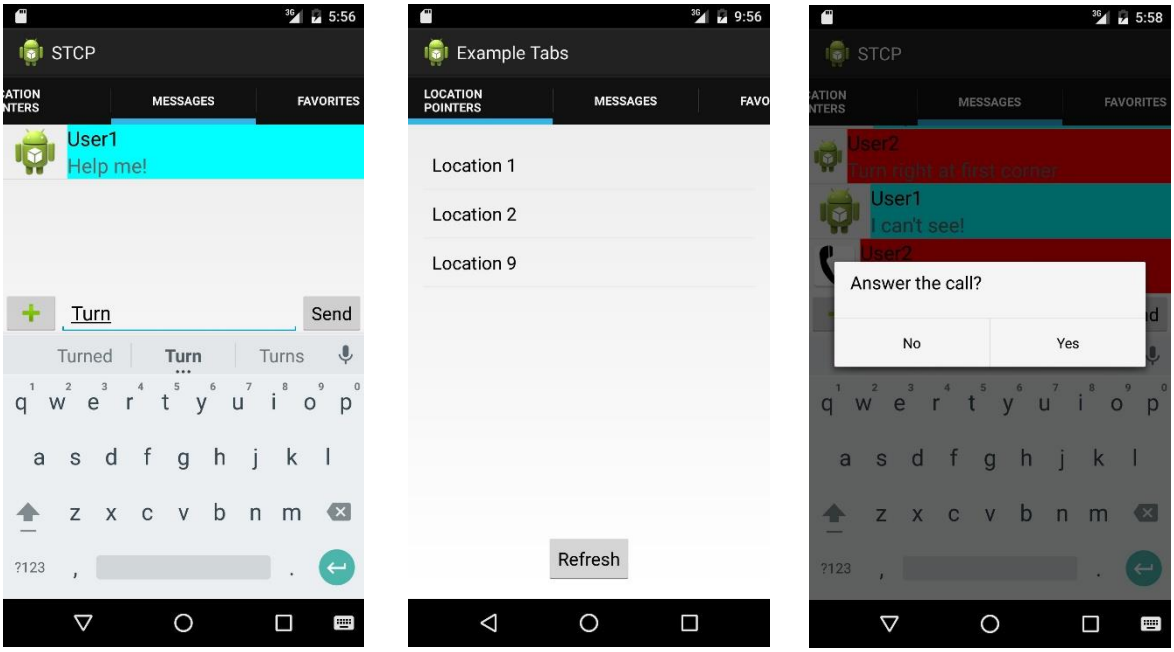
### 5.1 Server-side Implementation

Our server implementation consists of two major components: cloud-pointer server and cloud message server. All server components are implemented on a Linux machine running 3.14 kernel with 2 Intel Xeon E5-2650 2.0GHz octa-core CPU and 256GB RAM. The server components are mainly programmed by C++ and the

algorithm part related to cloud-pointers borrows the ability of MATLAB.

More specifically, the first component, cloud-pointer server, runs two modules: 1) fingerprint processor running on MATLAB, which clusters signal fingerprints as well as quantifies the similarity between fingerprints and 2) cloud-pointer manager that includes creation, deletion, and look-up of cloud-pointers as its key operations. The second component, cloud message server, is implemented with MySQL to store messages on the hash map of cloud-pointers with their signal fingerprints and accessible time period as their tags to be searched.

## 5.2 User-side Implementation




**(b) A user can send a message to the current location**

**(a) A user can see all the locations where she ever made a posting**

**(c) A user can see all the messages exchanged in a location and get call-backs**

**Figure 6: Our primitive SpaceMessaging user interface implemented on the Android platform.**

We implement user functions on various Android smartphones running from Kitkat to Marshmallow (4.1.2 ~ 6.0.1). Our implementation consists of three major components: space information collector, message handler, and tracking service. Space information collector regularly gathers WiFi signal fingerprint through the WiFi scan API. Message handler lets the user device display message contents appropriately according to their types when receiving messages. When sending message, message handler lets a user attach files or type in commands as well as plain texts. The type *f* of each messages is defined in the header of the message format, which can be either of *TXT*, *MEDIA*, *URL*, and *COMMAND*. TXT and MEDIA represent plain texts and multimedia data such as photos, audio files, and video files, respectively. URL means web-links that cover both hyperlinks and general web-queries following URI (Uniform Resource Identifier) originally proposed by [5]. COMMAND is a special message type that can be interpreted and executed by the message handler of its recipient. For instance, when COMMAND is given as *CALLBACK* with a phone number, it makes the recipient to give an automated call back to the phone number to initiate a direct voice channel between the message sender and the receiver. CALLBACK is able to not only initiate one-to-one voice call but also to create a conference call when the message sender wants to communicate with everybody associated with the given cloud-pointer. We implement these VoIP functions using an open-source library Linphone [13] and its SIP (session initiation protocol) server installed in our SpaceMessaging server. Note that COMMAND works only when its proper permission is preset in the receiving user device. Otherwise, COMMAND will be ignored at the receiver. Message handler also provides search function for stored messages via Android default *sqlite* library. In order to support all these features, message

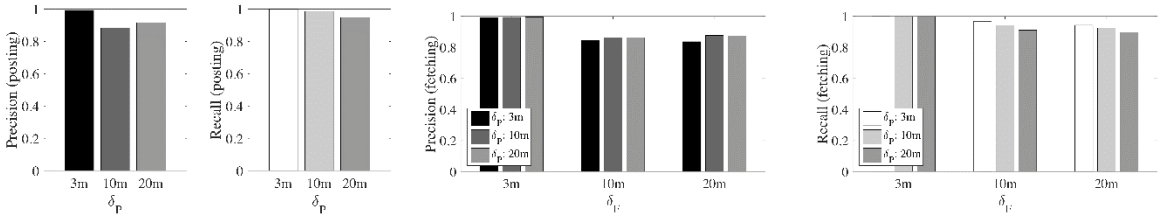
handler is implemented using Android content provider library. For the tracking service, we implement a GCM (Google Cloud Messaging) like keep-alive module, whose main feature is to report the IP address change of a user device to the tracking server, which typically happens when the device switches network interface (e.g., from LTE to WiFi) or moves to the coverage of a different access point. Our tracking service works essentially in the event-driven manner upon reception of the IP change intent and only consumes little energy.

The primitive version of user interface running on Android is depicted in Figure 6. As in Figure 6 (a), a user can simply send a message without knowing anything about signal fingerprint and will have the list of locations where she made postings as in Figure 6 (b). When she clicks into one of the locations, the device will fetch all the messages from the corresponding cloud-pointer and display the messages. If she sends a message with CALLBACK, she will be able to get a call from another user who is accessing the related cloud-pointer as shown in Figure 6 (c).

# 6 Evaluation

For the evaluation of our SpaceMessaging system, we extensively perform both trace-driven emulations as well as stress test to our Linux server. The trace-driven emulations based on about 19,000 fingerprint samples are specifically designed to evaluate the accuracy of message delivery and the stress test is to reveal the capacity bound of our server implementation.

## 6.1 Posting Accuracy




(a) Precision (posting) (b) Recall (posting) (c) Precision (fetchin) (d) Recall (fetching)

**Figure 7: The posting and fetching accuracy in the perspective of Precision and Recall from 19 cloud-pointers**

In order to analyze the SpaceMessaging accuracy, we emulate the posting and the fetching operations with a large amount of fingerprints that are sampled about 1000 times per each space from 19 total spaces depicted in Figure 2. We first formally define the events of *TP, TN, FP,* and *FN* for posting a message. With the help of the notion of $\delta_P$, *T* or *F* represent if the cloud-pointers to post a message satisfying the radius condition $\delta_P$ or not, respectively. *P* or *N* represent if the posting itself is made or not, respectively. Therefore, *TP* and *FP* stand for the events in which the posting is correctly made toward the cloud-pointers inside $\delta_P$ and is badly made toward the cloud-pointers outside $\delta_P$, respectively. Similarly, *TN* and *FN* stand for the events where the posting is not made toward the cloud-pointers outside $\delta_P$ which is welldone and the posting is not made toward the cloud-pointers inside $\delta_P$ which is of a mistake, respectively.

Now, we adopt two well-known statistical accuracy measures, *Precision* and *Recall* that are defined respectively as follows[2]:

$$Precision = \frac{TP}{(TP + FP)}, \quad (2)$$

$$Recall = \frac{TP}{(TP + FN)}, \quad (3)$$

[2] We have also evaluated F1 score which is the harmonic mean of *Precision* and *Recall,* but omitted those values as it can be immediately obtained from *Precision* and *Recall* values

where ⊷denotes the number of such an event.

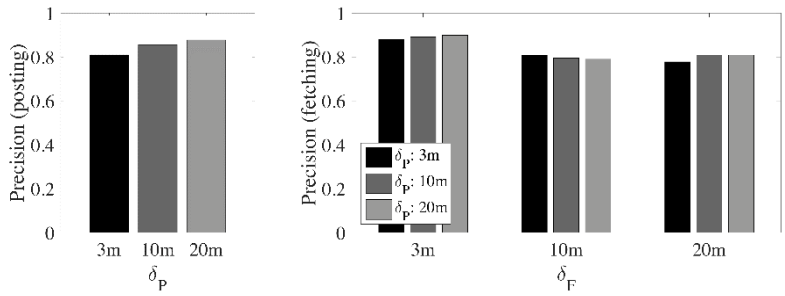
Therefore, by the definition, *Precision* quantifies how much portion of the postings are made within the radius condition $\delta_P$ and *Recall* assess how much portion of all the cloud-pointers within $\delta_P$ is covered by the postings. *Precision* and *Recall* are all important, but based on the objective of the system, more priority can be given toward either of those. If we take the disaster scene exemplified in the introduction as the main use case of SpaceMessaging, setting up the parameters of SpaceMessaging to minimize $1-Recall$ becomes the most urgent goal to achieve, where $1 - Recall$ is widely known as so called *Missrate*. In such a scenario, it is intuitively correct that missing out the delivery toward the people connected to some cloud-pointers within $\delta_P$ is more critical than the delivery is overly made even toward the people connected to the cloud-pointers outside $\delta_P$.

Running trace-driven emulations with the collected signal fingerprints whose 20% is devoted for training 19 clusters and the remaining 80% is used for evaluation, we get Figure 7 (a) and (b) as *Precision* and *Recall* for posting with various $\delta_P$ radii: 3m, 10m, and 20m. As shown in the figure, we tune the signal vector distance thresholds to maximize *Recall* for given $\delta_P$, thus obtaining at least 95% *Recall* for all $\delta_P$ values tested. Because of this tuning, *Precision* performance got slightly degraded by their intrinsic trade-off relationship. However, *Precision* still satisfies more than 88%, which is quite acceptable given a disaster scenario.

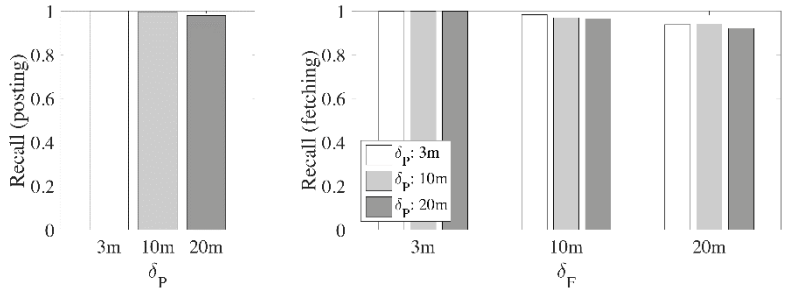
## 6.2 Fetching Accuracy

The fetching accuracy can be evaluated in a similar manner to the posting accuracy with a little more complication. We again define *TP, TN, FP,* and *FN* events for fetching. *TP* and *FP* for fetching stand for the events where the messages inside the target radius $\delta_F$ are correctly fetched and the messages outside $\delta_F$ are overly fetched. *TN* and *FN* stand for the events in which the messages outside $\delta_F$ are not fetched as intended and the messages inside $\delta_F$ are not fetched mistakenly. Therefore, for the evaluation of fetching accuracy, we need the messages already posted in the cloud-pointers. Given that posting with different $\delta_P$ values may distribute the messages in different ways, we evaluate the fetching accuracy from the combinations of ($\delta_P$,$\delta_F$). Figure 7 (c) and (d) shows *Precision* and *Recall* for fetching, when the training and testing are done in the same manner as in the posting accuracy evaluation. Note again that we have tuned the parameters to maximize *Recall*, as it is more important than *Precision* in the scenario we consider. Also note that if we need to consider a different scenario where *Precision* is more important than *Recall*, tuning the parameters to give a more favor to *Precision* is possible. As Figure 7 (c) and (d) show, even at the worst performing combination of ($\delta_P$,$\delta_F$) = (20m, 20m), we were able to achieve more than 90% of *Recall*, while the worst *Precision* remains beyond 83%.

## 6.3 Impact of Smaller Training Set

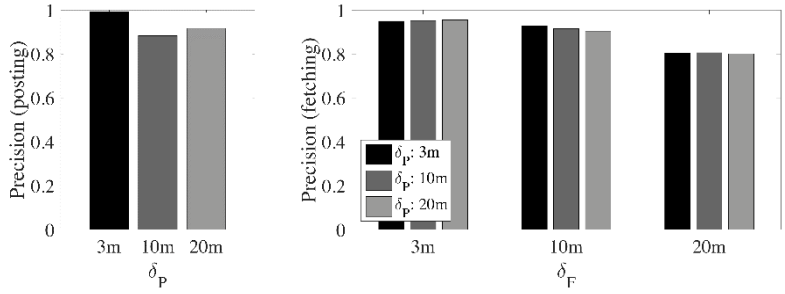



(a1) Precision(posting) (a2) Precision(fetching)

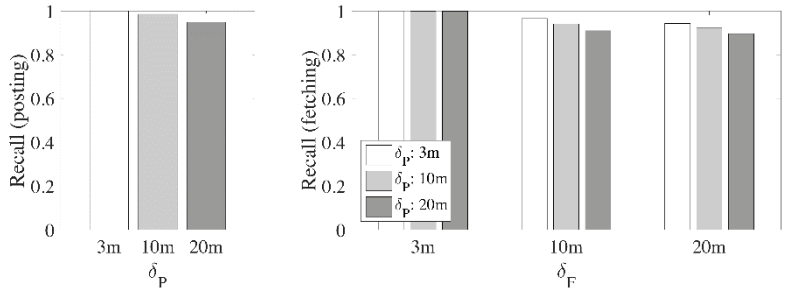



(a3) Recall(posting) (a4) Recall(fetching)

(a) The training sample is limited to be 25%




(b1) Precision(posting) (b2) Precision(fetching)




(b3) Recall(posting) (b4) Recall(fetching)

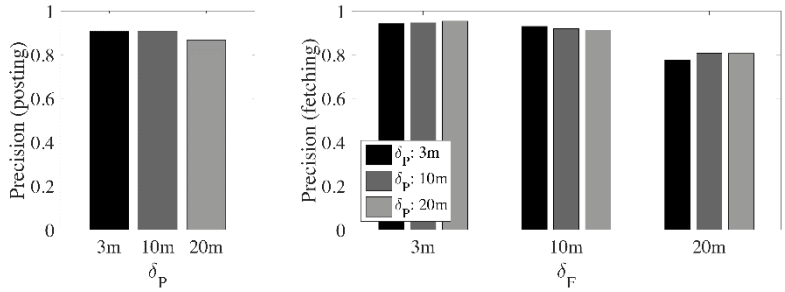
(b) The training sample is limited to be 60%

**Figure 8: The posting and fetching accuracy with limited fingerprint training samples.**

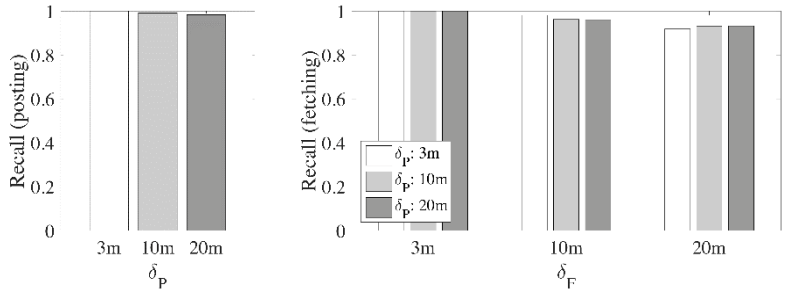
We evaluate the posting and fetching accuracy under the scenarios where the size of the training set is smaller. In the evaluation of Figure 7, the number of per-space fingerprint training samples used is about 200, which is 20% of the total. Now, we limit the training samples to be 25% and 60% of that for Figure 7 and show the performances in Figure 8 (a) and (b), respectively. As the Figure 8 shows, we find that *Precision* and *Recall* for

both posting and fetching are not substantially degraded. This result confirms that our fingerprint crowdsourcing process can be quick enough to provide reasonable performance to SpaceMessaging users.

## 6.4 Impact of Underdeveloped PoIs

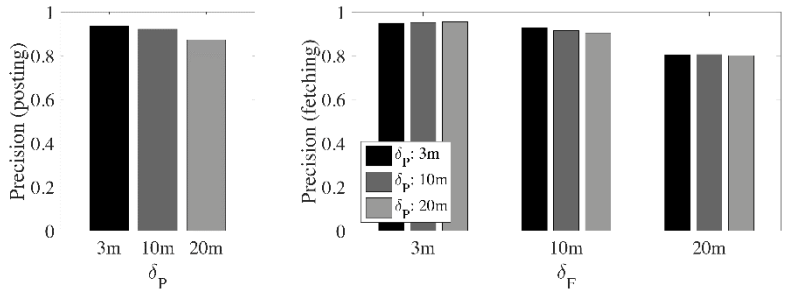



(a1) Precision(posting) (a2) Precision(fetching)

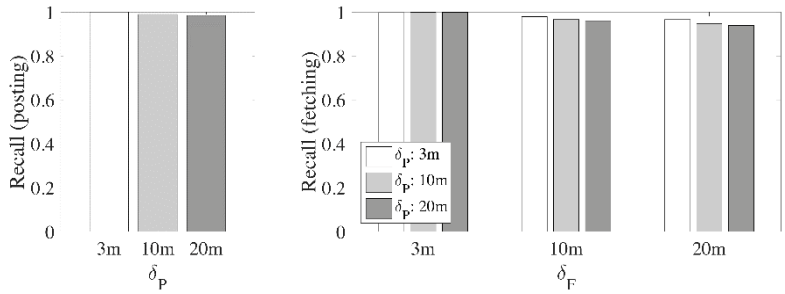



(a3) Recall(posting) (a4) Recall(fetching)

(a) The Performance when 12 actual PoIs are Clusterd by $k_{PoI}$ = 12




(b1) Precision(posting) (b2) Precision(fetching)




(b3) Recall(posting) (b4) Recall(fetching)

(b) The Performance when 12 actual PoIs are Clusterd by $k_{PoI}$ = 19

**Figure 9: The posting and fetching accuracy comparison between $k_{PoI}$ = 12 and $k_{PoI}$ = 19 where the actual number underlying PoIs is 12.**

We now evaluate the posting and fetching accuracy when there are fewer number of PoIs developed in a

building. This means that some of anticipated PoIs get relatively much less number of fingerprint samples compared to other developed PoIs. Immediate questions arising
in such an early stage of SpaceMessaging are: *can we still apply the anticipated number of clusters even though the actual number of clusters is far less than that?* and Are we not losing substantial performance from this mismatch? Figure 9 (a) and (b) show the posting and fetching accuracy when clustered with $k_{\text{PoI}} = 12$ and $k_{\text{PoI}} = 19$ under the actual number of PoIs in the building is only 12 instead of being 19. Figure 9 (a) and (b) which shows very similar performance confirm that thanks to the vector space distance thresholds set for $\delta_P$ and $\delta_F$, overly separating clusters is still safe to use.

# 7 Concluding Remarks

This work proposes a new kind of communication system, SpaceMessaging, which delivers messages to spaces hence not requiring any receiver identity information in advance. Using the cloud-pointers as a medium, which are virtually mapped to physical spaces, SpaceMessaging even enables communication between people who may not know each other. Our realization of SpaceMessaging relies on several techniques such as vectorization of signal fingerprints, clustering of such fingerprint vectors, and setting thresholds from vector distances. Through a set of extensive performance evaluations with implementations on Android and Linux systems, we validated that SpaceMessaging indeed works well and quantitatively evaluated that *Recall* can be maintained above 90% by tuning relevant parameters appropriately.

We have multiple direction to extend SpaceMessaging. Once is to develop more energyefficient server and mobile device architectures. Another is to exploit a wider range of sensory information other than WiFi signal fingerprints to improve the space pin-pointing ability. Launching an open source project to embed SpaceMessaging as an Android default service is also of our interest.

# References


[1] smslater.

[2] Ahn, K. H. Cell broadcasting service system and method. US Patent(2008).

[3] Bachir, A., and Benslimane, A. A multicast protocol in ad hoc networks inter-vehicle geocast. In *Proceeding of IEEE VTC 2003 Spring* (2003), vol. 4, pp. 2456–2460.

[4] Bahl, P., and Padmanabhan, V. N. Radar: An in-building rf-based user location and tracking system. In *In Proceeding of IEEE INFOCOM 2000* (2000), vol. 2, pp. 775–784.

[5] Berners-Lee, T. Universal resource identifiers in www.

[6] Buechner, M. M. Oh, don't forget send quick reminders, 2007.

[7] Capewell, M. autotext, Apr. 2013.

[8] Chen, Q., and Wang, B. Finccm: Fingerprint crowdsourcing, clustering and matching for indoor subarea localization. *IEEE Wireless Communications Letters 4*, 6 (2015), 677– 680.

[9] Huang, Q., Lu, C., and Roman, G.-C. Spatiotemporal multicast in sensor networks. In *Proceedings of ACM SenSys 2003* (2003), pp. 205–217.

[10] Jain, P., Manweiler, J., and Roy Choudhury, R. Overlay: Practical mobile augmented reality. In *Proceedings of ACM MobiSys 2015* (2015), pp. 331–344.

[11] Le Dortz, N., Gain, F., and Zetterberg, P. Wifi fingerprint indoor positioning system using probability distribution comparison. In *Proceedings of IEEE ICASSP 2012* (March 2012), pp. 2301–2304.

[12] Link, J. A. B., Smith, P., Viol, N., and Wehrle, K. Footpath: Accurate map-based indoor navigation using smartphones. In *Proceedings of IEEE IPIN 2011* (2011), pp. 1–8.

[13] Linphone.org. Linphone: Voip softphone - open source video sip phone, voip software, 2015.

[14] MacQueen, J. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (1967), vol. 1, Oakland, CA, USA., pp. 281–297.

[15] Navas, J. C., and Imielinski, T. Geocast—geographic addressing and routing. In *Proceedings of ACM/IEEE MobiCom 1997* (1997), pp. 66–76.

[16] Nilsson, M., Rantakokko, J., Skoglund, M., and Hendeby, G. Indoor positioning using multi-frequency rss with foot-mounted ins. In *Proceedings of IEEE IPIN 2014* (Oct 2014), pp. 177–186.

[17] OpenSignal. The state of lte february 2016, 2016.

[18] Places.foursquare.com. Places by foursquare, 2015.

[19] Rai, A., Chintalapudi, K. K., Padmanabhan, V. N., and Sen, R. Zee: zero-effort crowdsourcing for indoor localization. In *Proceedings of ACM MobiCom 2012* (2012), pp. 293–304.

[20] Rai, A., Chintalapudi, K. K., Padmanabhan, V. N., and Sen, R. Zee: Zero-effort crowdsourcing for indoor localization. In *Proceedings of ACM MobiCom 2012* (2012), pp. 293–304.

[21] Rodoplu, V., and Meng, T. H. Minimum energy mobile wireless networks. *IEEE Journal on Selected Areas in Communications 17*, 8 (1999), 1333–1344.

[22] Singh, S., and Agrawal, S. Vanet routing protocols: Issues and challenges. In *Engineering and Computational Sciences (RAECS), 2014 Recent Advances in* (March 2014), pp. 1–5.

[23] Wang, H., Sen, S., Elgohary, A., Farid, M., Youssef, M., and Choudhury, R. R. No need to war-drive: Unsupervised indoor localization. In *Proceedings of ACM MobiSys 2012* (2012), pp. 197–210.

[24] Ward Jr, J. H. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association 58*, 301 (1963), 236–244.

[25] Yang, Z., Wu, C., and Liu, Y. Locating in fingerprint space: Wireless indoor localization with little human intervention. In *Proceedings of ACM MobiCom 2012* (2012), pp. 269–280.